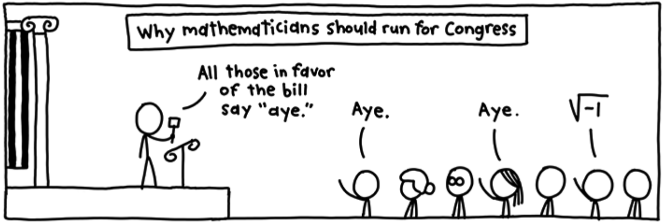
# [1] The Field

# The Field: Introduction to complex numbers

Solutions to $x^2 = -1$?

Mathematicians invented $\mathbf{i}$ to be one solution




Guest Week: Bill Amend (excerpt, `http://xkcd.com/824`)

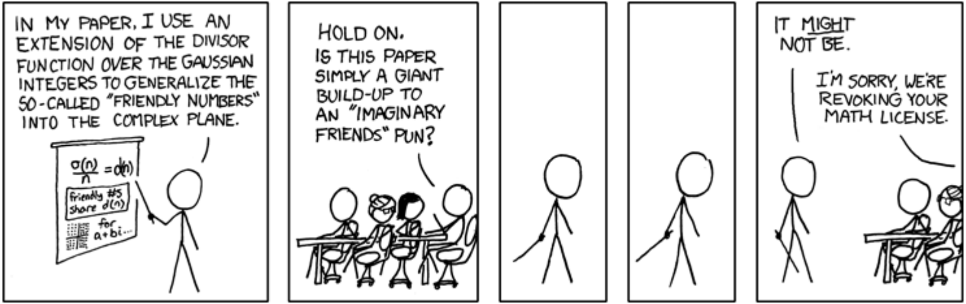
Can use $\mathbf{i}$ to solve other equations, e.g.:

$$x^2 = -9$$

Solution is $x = 3\mathbf{i}$

# Introduction to complex numbers

Numbers such as $\mathbf{i}$, $-\mathbf{i}$, $3\mathbf{i}, 2.17\mathbf{i}$ are called *imaginary* numbers.




Math Paper (`http://xkcd.com/410`)

# The Field: Introduction to complex numbers

- Solution to $(x-1)^2 = -9$?
- One is $x = 1 + 3\mathbf{i}$.

- A real number plus an imaginary number is a *complex number*.
- A complex number has a *real part* and an *imaginary part*.

$$\text{complex number} = (\text{real part}) + (\text{imaginary part})\,\mathbf{i}$$

# The Field: Complex numbers in Python

# Abstracting over *Fields*

- *Overloading:* Same names (+, etc.) used in Python for operations on real numbers and for operations complex numbers
- Write procedure `solve(a,b,c)` to solve $ax + b = c$:

  ```
  >>> def solve(a,b,c): return (c-b)/a
  ```

  Can now solve equation $10x + 5 = 30$:

  ```
  >>> solve(10, 5, 30)
  2.5
  ```

- Can also solve equation $(10 + 5\mathbf{i})x + 5 = 20$:

  ```
  >>> solve(10+5j, 5, 20)
  (1.2-0.6j)
  ```

- Same procedure works on complex numbers.

# Abstracting over *Fields*

Why does procedure works with complex numbers?

Correctness based on:

- / is inverse of *
- - is inverse of +

Similarly, much of linear algebra based just on +, -, *, / and algebraic properties

- / is inverse of *
- - is inverse of +
- *addition is commutative:* $a + b = b + a$
- *multiplication distributes over addition:* $a * (b + c) = a * b + a * c$
- etc.

You can plug in any collection of "numbers" with arithmetic operators +, -, *, / satisfying the algebraic properties— and much of linear algebra will still "work".

Such a collection of "numbers" with +, -, *, / is called a *field*.
Different fields are like different classes obeying the same interface.

# Field notation

When we want to refer to a field without specifying which field, we will use the notation $\mathbb{F}$.

# Abstracting over *Fields*

We study three fields:

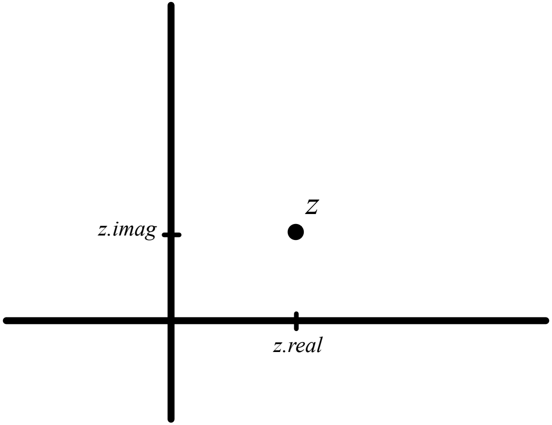
- The field $\mathbb{R}$ of real numbers
- The field $\mathbb{C}$ of complex numbers
- The finite field $GF(2)$, which consists of 0 and 1 under mod 2 arithmetic.

Reasons for studying the field $\mathbb{C}$ of complex numbers:

- $\mathbb{C}$ is similar enough to $\mathbb{R}$ to be familiar but different enough to illustrate the idea of a field.
- Complex numbers are built into Python.
- Complex numbers are the intellectual ancestors of vectors.
- In more advanced parts of linear algebra (to be covered in a follow-on course), complex numbers play an important role.

# Complex numbers as points in the complex plane

Can interpret *real* and *imaginary* parts of a complex number as $x$ and $y$ coordinates.
Thus can interpret a complex number as a *point* in the plane

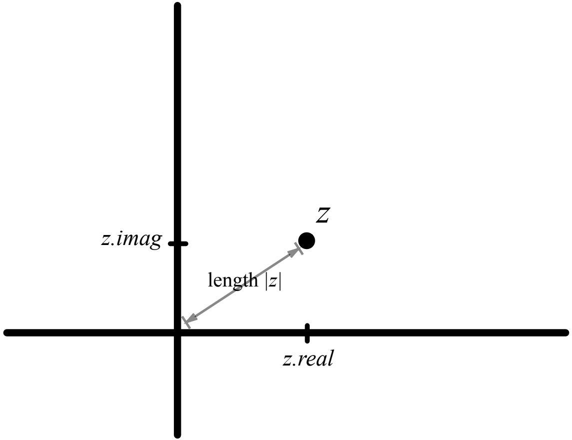



(the *complex plane*)

# Playing with $\mathbb{C}$

# Playing with $\mathbb{C}$: The absolute value of a complex number

*Absolute value* of $z$ = distance from the origin to the point $z$ in the complex plane.

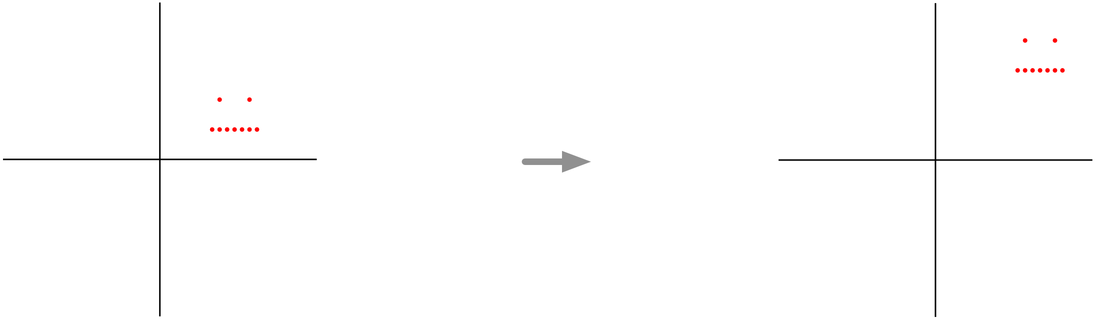



- In Mathese, written $|z|$.
- In Python, written `abs(z)`.

# Playing with $\mathbb{C}$: Adding complex numbers

Geometric interpretation of $f(z) = z + (1 + 2\mathbf{i})$?

Increase each real coordinate by 1 and increases each imaginary coordinate by 2.



$f(z) = z + (1 + 2\mathbf{i})$ is called a *translation*.

# Playing with $\mathbb{C}$: Adding complex numbers

- *Translation in general:*

  $$f(z) = z + z_0$$

  where $z_0$ is a complex number.

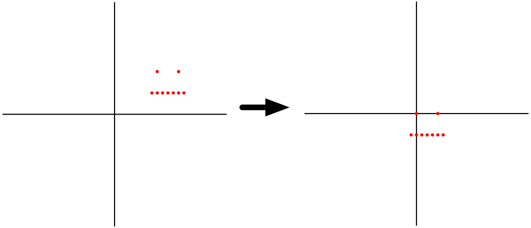
- A translation can "move" the picture anywhere in the complex plane.

# Playing with $\mathbb{C}$: Adding complex numbers

- *Quiz:* The "left eye" of the list $L$ of complex numbers is located at $2 + 2\mathbf{i}$. For what complex number $z_0$ does the translation

$$f(z) = z + z_0$$

move the left eye to the origin $0 + 0\mathbf{i}$?

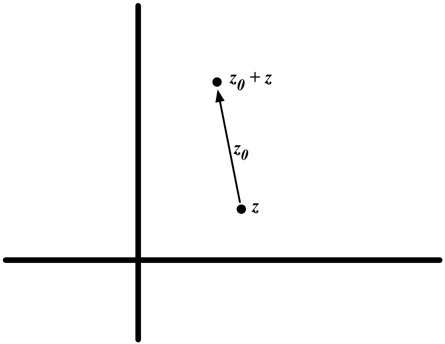


- *Answer:* $z_0 = -2 - 2\mathbf{i}$

# Playing with $\mathbb{C}$: Adding complex numbers: Complex numbers as arrows

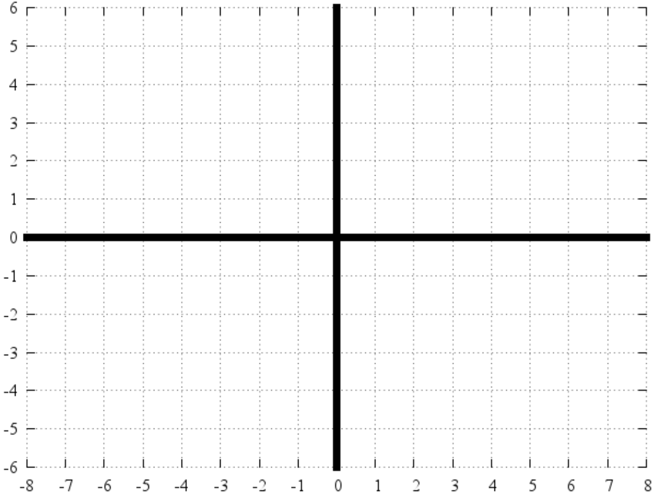
Interpret $z_0$ as representing the translation $f(z) = z + z_0$.

- Visualize a complex number $z_0$ as an arrow.
- Arrow's tail located an any point $z$
- Arrow's head located at $z + z_0$
- Shows an example of what the translation $f(z) = z + z_0$ does



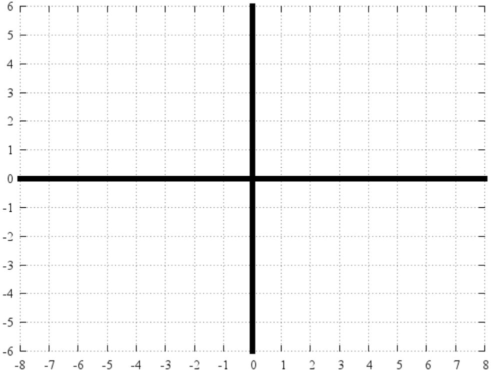
# Playing with $\mathbb{C}$: Adding complex numbers: Complex numbers as arrows

*Example:* Represent $-6 + 5\mathbf{i}$ as an arrow.



# Playing with $\mathbb{C}$: Adding complex numbers: Composing translations, adding arrows

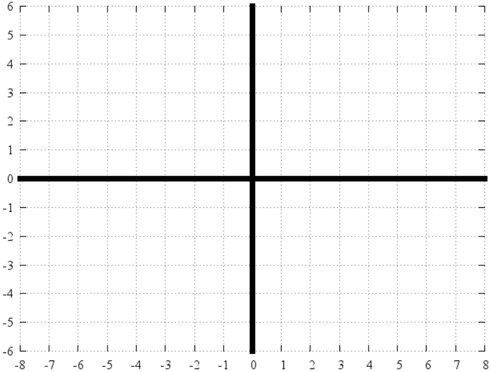
- Consider two complex numbers $z_1$ and $z_2$.
- They correspond to translations $f_1(z) = z + z_1$ and $f_2(z) = z + z_2$
- Functional composition: $(f_1 \circ f_2)(z) = z + z_1 + z_2$
- Represent functional composition by adding arrows.
- *Example:* $z_1 = 2 + 3\mathbf{i}$ and $z_2 = 3 + 1\mathbf{i}$



# Playing with $\mathbb{C}$: Multiplying complex numbers by a positive real number

Multiply each complex number by 0.5

$f(z) = 0.5\,z$



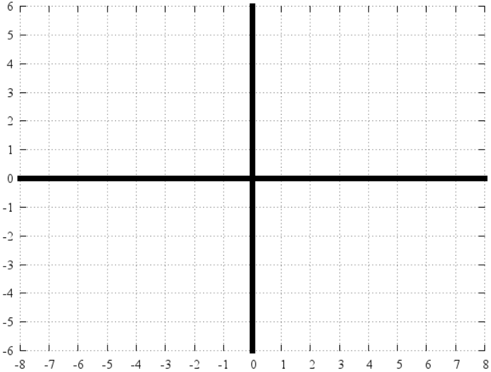

Arrow in same direction but half the length.

$\boxed{\textit{Scaling}}$

# Playing with $\mathbb{C}$: Multiplying complex numbers by a negative number

$f(z) = (-1)\,z$

Multiply each complex number by -1




Arrow in opposite direction

$\boxed{\textit{Rotation by 180 degrees}}$

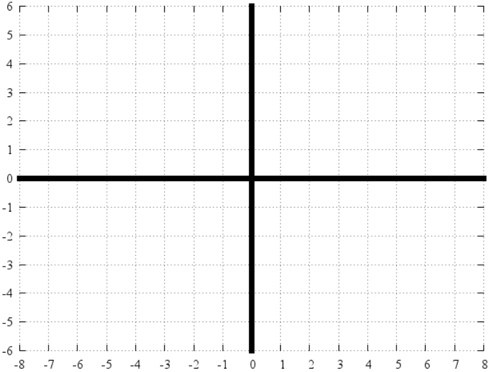
# Playing with $\mathbb{C}$: Multiplying by $\mathbf{i}$: rotation by 90 degrees

How to rotate counterclockwise by $90°$?

Need $x + y\,\mathbf{i} \mapsto -y + x\,\mathbf{i}$

Use $\mathbf{i}(x + y\mathbf{i}) = x\,\mathbf{i} + y\,\mathbf{i}^2 = x\,\mathbf{i} - y$

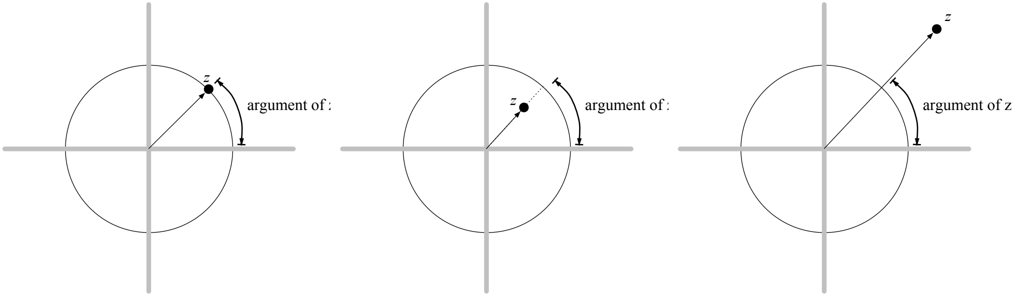
$f(z) = \mathbf{i}\,z$



# Playing with $\mathbb{C}$: The unit circle in the complex plane: *argument* and angle

What about rotating by another angle?

**Definition:** *Argument* of $z$ is the angle in radians between $z$ arrow and $1 + 0\mathbf{i}$ arrow.




$\boxed{\text{Rotating a complex number } z \text{ means } \textit{increasing its argument}.}$

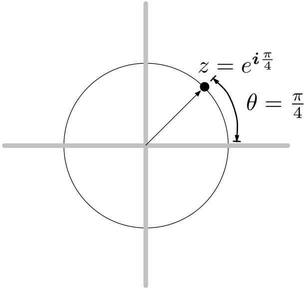
# Playing with $\mathbb{C}$: Euler's formula

"He calculated just as men breathe, as eagles sustain themselves in the air."
*Said of Leonhard Euler*

**Euler's formula:** For any real number $\theta$,

$$e^{\theta\,\mathbf{i}}$$

is the point $z$ on the unit circle with argument $\theta$.




$e = 2.718281828\ldots$

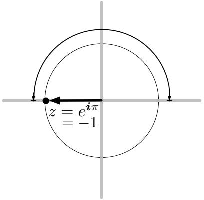
# Playing with $\mathbb{C}$: Euler's formula

**Euler's formula:** For any real number $\theta$,

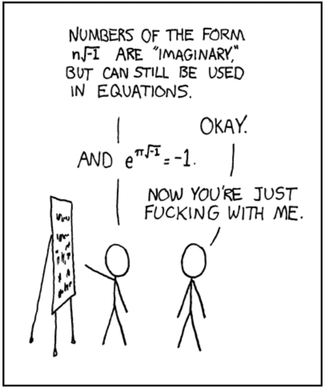
$$e^{\theta \mathbf{i}}$$

is the point $z$ on the unit circle with argument $\theta$.

Plug in $\theta = \pi$....







## Playing with $\mathbb{C}$: Euler's formula

Plot

$$e^{0\cdot\frac{2\pi\mathbf{i}}{20}},\ e^{1\cdot\frac{2\pi\mathbf{i}}{20}}, e^{2\cdot\frac{2\pi\mathbf{i}}{20}}, e^{3\cdot\frac{2\pi\mathbf{i}}{20}}, \ldots, e^{19\cdot\frac{2\pi\mathbf{i}}{20}}$$

# Playing with $\mathbb{C}$: Rotation by $\tau$ radians

Back to question of rotation by any angle $\tau$.

- Every complex number can be written in the form $z = re^{\theta\mathbf{i}}$
  - $r$ is the absolute value of $z$
  - $\theta$ is the argument of $z$
- Need to increase the argument of $z$
- Use exponentiation law $e^a \cdot e^b = e^{a+b}$
- $re^{\theta\mathbf{i}} \cdot e^{\tau\mathbf{i}} = re^{\theta\mathbf{i}+\tau\mathbf{i}} = re^{(\theta+\tau)\mathbf{i}}$
- $f(z) = z \cdot e^{\tau\mathbf{i}}$ does rotation by angle $\tau$.

# Playing with $\mathbb{C}$: Rotation by $\tau$ radians

Rotation by $3\pi/4$

# Playing with $GF(2)$

Galois Field 2

has just two elements: 0 and 1

Addition is like exclusive-or:

| $+$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Multiplication is like ordinary multiplication

| $\times$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Evariste Galois, 1811-1832

Usual algebraic laws still hold, e.g. multiplication distributes over addition

$a \cdot (b + c) = a \cdot b + a \cdot c$

# $GF(2)$ in Python

We provide a module `GF2` that defines a value `one`.
This value acts like 1 in $GF(2)$:

```
>>> from GF2 import one
>>> one + one
0
>>> one * one
one
>>> one * 0
0
>>> one/one
one
```

We will use `one` in coding with $GF(2)$.

# Playing with $GF(2)$: Encryption

Alice wants to arrange with Bob to communicate one bit $p$ (the *plaintext*).
To ensure privacy, they use a cryptosystem:

- Alice and Bob agree beforehand on a secret key $k$.
- Alice encrypts the plaintext $p$ using the key $k$, obtaining the cyphertext $c$ according to the table

| $p$ | $k$ | $c$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Q:** Can Bob uniquely decrypt the cyphertext?
**A:** Yes: for any value of $k$ and any value of $c$, there is just one consistent value for $p$.

An eavesdropper, Eve, observes the value of $c$ (but does not know the key $k$).
**Question:** Does Eve learn anything about the value of $p$?
**Simple answer:** No:

- if $c = 0$, Eve doesn't know if $p = 0$ or $p = 1$ (both are consistent with $c = 0$).
- if $c = 1$, Eve doesn't know if $p = 0$ or $p = 1$ (both are consistent with $c = 1$).

**More sophisticated answer:** It depends on how the secret key $k$ is chosen.
Suppose $k$ is chosen by flipping a coin:

Probability is $\frac{1}{2}$ that $k = 0$

# Playing with $GF(2)$: One-to-one and onto function and perfect secrecy

| $p$ | $k$ | $c$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

What is it about this cryptosystem that leads to perfect secrecy? Why does Eve learn nothing from eavesdropping?

Define $f_0 : GF(2) \longrightarrow GF(2)$ by

$f_0(k) =$encryption of $p = 0$ with key $k$

According to the first two rows of the table,

$$f_0(0) = 0 \text{ and } f_0(1) = 1$$

$\boxed{\text{This function is one-to-one and onto.}}$

When key $k$ is chosen uniformly at random

$$\text{Prob}[k = 0] = \frac{1}{2}, \text{Prob}[k = 1] = \frac{1}{2}$$

the probability distribution of the output $f_0(k) = p$ is also uniform:

$$\text{Prob}[f_0(k) = 0] = \frac{1}{2}, \text{Prob}[f_0(k) = 1] = \frac{1}{2}$$

Define $f_1 : GF(2) \longrightarrow GF(2)$ by

$f_1(k) =$encryption of $p = 1$ with key $k$

According to the last two rows of the table,

$$f_1(0) = 1 \text{ and } f_1(1) = 0$$

$\boxed{\text{This function is one-to-one and onto.}}$

When key $k$ is chosen uniformly at random

$$\text{Prob}[k = 0] = \frac{1}{2}, \text{Prob}[k = 1] = \frac{1}{2}$$

the probability distribution of the output $f_1(k) = p$ is also uniform:

$$\text{Prob}[f_1(k) = 1] = \frac{1}{2}, \text{Prob}[f_1(k) = 0] = \frac{1}{2}$$

The probability distribution of the cyphertext does not depend on the plaintext!

# Perfect secrecy

| $p$ | $k$ | $c$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Idea is the basis for cryptosystem: the **one-time pad**.

If each bit is encrypted with its own one-bit key, the cryptosystem is unbreakable

In the 1940's the Soviets started re-using bits of key that had already been used.

Unfortunately for them, this was discovered by the US Army's Signal Intelligence Service in the top-secret VENONA project.

This led to a tiny but historically significant portion of the Soviet traffic being cracked, including intelligence on

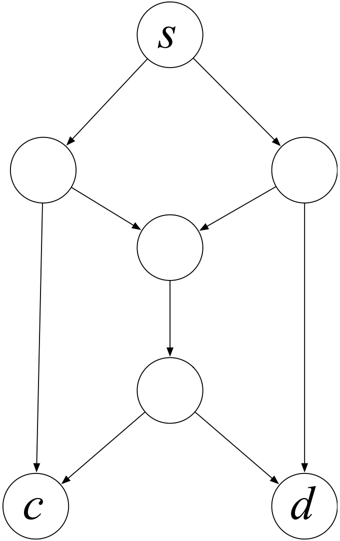
- spies such as Julius Rosenberg and Donald Maclean, and
- Soviet espionage on US technology including nuclear weapons.

The public only learned of VENONA when it was declassified in 1995.

# Playing with $GF(2)$: Network coding

Streaming video through a network

- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention
- Network coding doubles throughput in this example!

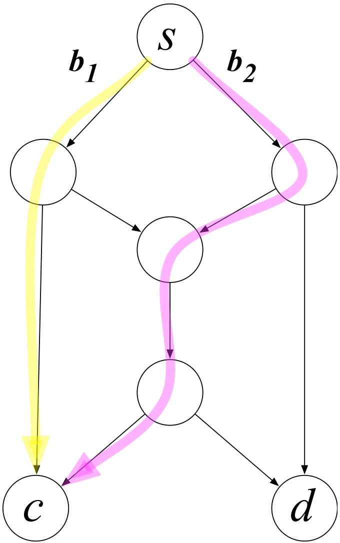


# Playing with $GF(2)$: Network coding

Streaming video through a network

- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention

- Network coding doubles throughput in this example!

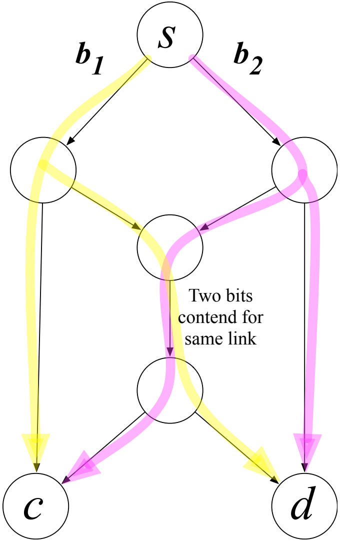


# Playing with $GF(2)$: Network coding

Streaming video through a network

- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention

- Network coding doubles throughput in this example!



# Playing with $GF(2)$: Network coding

Streaming video through a network

- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention
- Network coding doubles throughput in this example!